AD-A060 213 CALIFORNIA UNIV IRVINE DEPT OF INFORMATION AND COMP--ETC F/G 9/2
ANALYSIS OF A PROTOCOL USING A TOKEN FLOW MODEL.(U)
MAY 76 K LARSON MDA903-75-G-0001
UNCLASSIFIED UCI-ICS-TR-83 NL

| OF |
AD
A060 213

END
DATE
FILMED
12-78
DDC

ICS

UCCI

ICS

UCCI

ICS

Analysis of a Protocol Using a Token Flow Model

Ken Larson

Technical Report #83

University of California

Irvine

May 1, 1976

This work has been supported by the Advanced Research Projects Agency of the Department of Defense under Grant MDA903-75-G-0001.

Analysis of a Protocol Using a Token Flow Model

This report describes the analysis of a communications protocol using a token flow model. The protocol under study is designed to achieve computer network security within the communications system[1,2]. In the first section, the protocol is briefly described. In the second section, the model of the protocol is introduced. The third section presents the results generated when the analysis is applied to the model of the protocol.

PROTOCOL

The protocol is based on a scheme utilizing dynamic naming of processes. I will describe the protocol at a very high level, introducing only those aspects of the protocol, specifically message traffic, which are salient to the model. The protocol achieves security by using a sequence of message destination names for a single process and is best introduced by an example. Two processes ODD and EVEN are to communicate. For convenience ODD will use the sequence of names 1 and 3, and EVEN will use the sequence of names 0,2 and 4. ODD begins the session using the receive name(RN) 1; EVEN uses the receive name 0 to start. ODD begins the communication by sending a message to EVEN using as the destination address the initial receive name of EVEN, 0. The message includes a return address which gives the current receive name of ODD, 1. When EVEN receives the message addressed to its RN, 0, the process EVEN changes its receive name to 2. For reasons described later, EVEN retains 0 as the old receive name(ORN). EVFN then returns a message to ODD using the return address, 1, as the message destination address and using EVEN's current RN, 2, as the message return address. When ODD receives the message, it changes its RN to 3 retaining 1 as the ORN. It returns a message to EVEN using the return address of the received message, 2, as the message destination address, and its RN, 3, as the message return address. When EVEN receives the message with destination address 2, it changes its RN to 4, retains the ORN, 0, as the old-old receive name(OORN), and retains 2 as the ORN. EVEN then returns a message to ODD. Theinterprocess communication continues in this synchronous fashion. In practice, the names are generated by pseudo-random number generators. The protocol can operate if each process uses only two names in a cyclic way, that is, EVEN can use 2 then 4 then 2, and ODD can use 1 then 3 then 1.

A means is provided for recovery when a message is lost. There are three types of messages used to reestablish synchronization: reset, reset-response, and ok messages. Assume a message with destination address 3 and return address 4 is sent to the process ODD which has RN 3 from EVEN which has receive name 4. If this message is lost, both ODD and EVEN are waiting for messages, but it is EVEN who must supply a copy of the lost message. Both ODD and EVEN at some time experience a timeout waiting for the messages. In the same way that the RN is retained as the ORN, the names used to transmit messages, the destination names, are retained as the old transmit name(OTN). When a process fails to receive a message and a timeout occurs, the process sends a reset message using the OTN as the destination address. ODD sends a reset message to EVEN using OTN 0 as the destination address, since the last message was sent to 2, not the current RN of EVEN, 4. EVEN sends a reset message to ODD using the OTN, 1, since the last message, the one that was lost, was sent to 3(the current TN of EVEN). EVEN, with RN 4, recognizes that the reset message is destined for the OORN,0, and returns an ok message to ODD using the OTN, 1, as the destination address. This indicates to ODD that the last message from ODD was received. ODD receives the reset message to its ORN,1, and returns a reset-response message using OTN as the destination address. This indicates to EVEN that the last message it sent was not received. If in the intercum the message was received, ODD would send an ok to EVEN, using the OTN as the destination address.

If EVEN receives an ok, its message was received and it waits for a normal message. If EVEN receives a reset-response, the last message it sent was lost and EVEN retransmits the lost message. If any of the re-synchronization messages are lost, it is assumed that there is a major network failure, the connection is broken, and both processes return to the initial state.

MODEL

The protocol is modelled as a state machine. The state of the system is a column vector, S, whose elements are non-negative integers and represent either boolean (0 or 1) state information or the number of messages of a specific type present in the system. The protocol is represented in the model as a set of transitions which specify rules for state changes in the column vector. For convenience the elements of S have names called tokens. If S[5] is the element associated with a token X, then we write S[X] for convenience, if no confusion results. The number of tokens of type X is given by S[X]. The elements of the state vector represent three types of information. First, the state of a specific protocol process is represented by the existence in the system state of one of the tokens send(rn), wm(rn), or wr(rn). Second, the existence of a normal message, reset message, reset-response message or ok message to destination address x is represented by msg(x), reset(x), rr(x), or ok(x) respectively. For convenience, the receive names cycle between 0 and 2 for EVEN and 1 and 3 for ODD.

Figure 1 shows the template of transitions used to generate a model of the protocol. A transition is interpreted as a rule that states that whenever all tokens on the left-hand-side of the rule are present in S (the associated elements of S are greater than zero), then they can be removed from S (the appropriate elements of S decremented) and replaced by the tokens on the right-hand-side of the rule (the appropriate elements in S incremented by 1). If two transitions are possible, there is no assumption made about ordering, but they cannot occur simultaneously. The decrementing and incrementing of the elements of S as a transtion occurs can be represented by a vector t whose elements are 1 for an increment of an element of S and -1 for a decrement of an element of S and 0 for no change. The new state s′ after the transition t is given by

$$s' = s + t.$$

Since the tokens must be present for the transition to

```
T1: send(n) -> msg(n-1) & wm(n) & r(n)  [send message]

T2: r(n) -> reset(n-3) & wr(n)          [timeout]

T3: wm(n) & msg(n) & r(n) -> send(n+2)  [message recieved]

Response to reset:

T4: wm(n) & reset(n-4) -> ok(n-3) & wm(n)

                                   [your message  was received]

T5: wm(n) & lm(n) & reset(n-2) -> rr(n-1) & wm(n)

                                   [please retransmit]

Response to rr/ok:

T6: wr(n) & ok(n-4) -> r(n)        [my message was received]

T7: wr(n) & ok(n-2) -> r(n)        [my message was recieved]

T8: wr(n) & rr(n-2) -> r(n) & msg(n-1)  [retransmit]

Error transition:

T9: msg(n) -> lm(n)                [message lost]
```

If rn=n then oorn=n-4, orn=n-2, tn=n-1, otn=n-3, and nrn=n+2.

Figure 1. Transition Rules

occur, then $s' > 0$ always. To generate a specific instance of a protocol process an assignment is necessary for n. For example, the assignment in which EVEN has n=2 is the instance of the protocol process EVEN with receive name 2. The total model consists of the union of the transitions generated by each assignment cooresponding to a specific instance of a process receive name to be modelled. In the model presented, there are four assignments corresponding to EVEN with receive name(RN) 0, EVEN with receive name 2, ODD with receive name 1, and ODD with receive name 3.

ANALYSIS

By analyzing the transition rules a set of conservation rules can be generated. These hold for any system state which can be reached from the initial state d, using the transition rules. The initial state d is defined by d[send(1)]=1, d[wm(0)]=1, and d[r(0)]=1. If s is any state reached from d by a sequence of transition vectors, then (s-d) is in the vector space spanned by the transition vectors, i.e. (s-d) can be written as a sum of transition vectors. If possible, construct a vector q which is orthogonal to all the transition vectors t, i.e. the inner product of q with all the transition vectors is zero. If s is a state reached from d by a sequence of transitions, the inner product of (s-d) and q is zero since (s-d) can be written as a sum of transition vectors. Using Gaussian reduction, or a similar technique, a set of such vectors can be found and a set of conservation equations generated which the possible vectors (s-d) must satisfy. These token conservation rules are given in Figure 2. The conservation rules show that the number of tokens in the system state is bounded, since no token can appear more than once. Equations (1) and (2) of Figure 2 show that each process must be in one and only one "normal" state. Equations (3) and (4) show that each process can be in only one "reset" state. Equation (5) insures that either a process is ready to send, or there is a message, or there is a lost message, or there is a reset-response resulting from the lost message. Equations (6) and (7) show that either a process is ready to send a reset when a timeout is received (r), or it has sent a reset, or it has received a response (rr or ok).

Given these contraints, we would like to know if the protocol can halt, since we expect it to cycle through the synchronous transmission sequence indefinitely. The protocol will halt if it can reach a state, S, in which no transition vector can be applied without causing an element of S to be negative. We can write this condition as a set

```
C1: S[send(1,3)] + S[wm(1,3)] = 1

C2: S[send(0,2)] + S[wm(0,2)] = 1

C3: S[r(1,3)] + S[wr(1,3)] + S[send(1,3)] = 1

C4: S[r(0,2)] + S[wr(0,2)] + S[send(0,2)] = 1

C5: S[send(0:3)] + S[msg(0:3)] + S[lm(0:3)] + S[rr(0:3)] = 1

C6: S[send(0,2)] + S[r(0,2)] + S[reset(1,3)] + S[rr(0,2)] +
                                            S[ok(0,2)] = 1

C7: S[send(1,3)] + S[r(1,3)] + S[reset(0,2)] + S[rr(1,3)] +
                                            S[ok(1,3)] = 1


where

S[x(a,b)] = S[x(a)] + S[x(b)],

and

S[x(0:3)] = S[x(0)] + S[x(1)] + S[x(2)] + S[x(3)].
```

Figure 2. Conservation Rules

of linear contraints as shown in Figure 3.

In reducing the matrix of transition vectors to derive the contraint equations of Figure 2, information about the the possible states which can be reached has been lost. We know a msg must exist, but which one is it. This sequencing information can be generated by analyzing graphs which represent the. ordering of the transitions. A constraint graph, G1, G2,...G7 is constructed for each constraint equation. The vertices of the constraint graph are labelled with the names of the tokens which appear in the constraint equation. An edge (a,b) exists in the graph if there is a transition rule in which a appears on the left-hand-side and b appears on the right-hand-side. The constraint equations guarantee that for each state, S, of a state sequence there is one and only one vertex, v, in any constraint graph Gi such that S[v]=1. Futher, by the construction of the graph, for any state sequence there exists a corresponding path in each of the constraint graphs such that the i-th element of the path is a vertex, v, such that S[v]=1, where S is the i-th state of the state sequence. Since vertices with the same label can exist in more than one graph, there are states of the state sequence at which paths in the separate constraint graphs must coincide. This gives rise to explicit conservation rules. Small sections of constraint

H1: $S[send(n)] = 0$

H2: $S[r(n)] = 0$

H3: $S[wm(n)] + S[msg(n)] + S[r(n)] \leq 2$

H4: $S[wm(n)] + S[reset(n-4)] \leq 1$

H5: $S[wm(n)] + S[lm(n)] + S[reset(n-2)] \leq 2$

H6: $S[wr(n)] + S[ok(n-4)] \leq 1$

H7: $S[wr(n)] + S[ok(n-2)] \leq 1$

H8: $S[wr(n)] + S[rr(n-2)] \leq 1$

H9: $S[msg(n)] = 0$

For $n = 0,1,2,3$ (all numbers modulo four).

Figure 3. Halting Conditions

graphs G1 and G5 are shown in Figure 4a. The correspondnce shown for these sections exist in the other sections and lead to the constraint equation

C5a: S[wm(n)] = S[msg(n-1)]+S[msg(n)]+S[lm(n-1)]+S[lm(n)]+
S[send(n+1)]+S[rr(n-2)]+S[rr(n-1)].

The same techniques can be applied, as shown in Figure 4b, to show that

C6/7a: S[wr(n)] = S[reset(n-3)]+S[ok(n-2)]+S[rr(n-2)].

Combining the revised constraint equations and the halting conditions yeilds a single system of simultaneous inequalities and equalities. By using conditions H1 and H2 and the constraint equations C1-4, it can be shown that there are integers k and n such that S[wm(n)]=1, S[wm(k)]=1,S[wr(n)]=1, and S[wr(k)]=1. Using H4 implies that S[reset(n-4)]=0 and S[reset(k-4)]=0. From H7 and 8 we see that S[ok(n-2)]=0, S[ok(k-2)]=0, S[rr(k-2)]=0, and S[rr(n-2)]=0. But by the revised constraint C6/7a,

S[wm(n)] = S[ok(n-2)]+S[rr(n-2)]+S[reset(n-3)] = 1,

and

S[wm(k)] = S[ok(k-2)]+S[rr(k-2)]+S[reset(k-3)] = 1.

Therefore, S[reset(n-3)]=1 and S[reset(k-3)]=1. Using H5

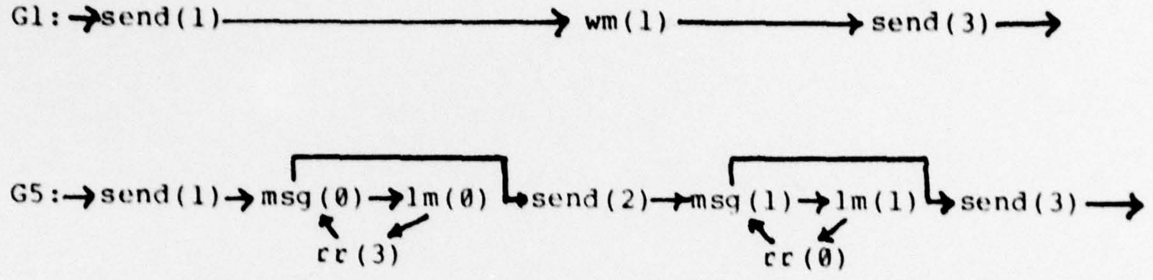

Figure 4a. Constraint Graphs G1 and G5

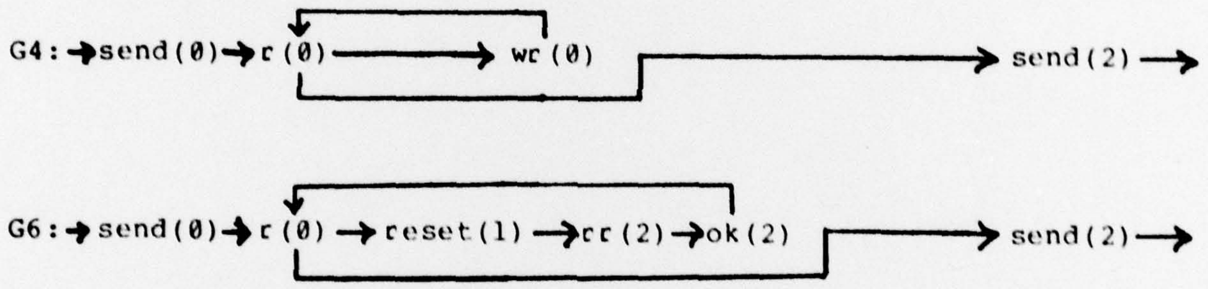

Figure 4b. Constraint Graphs G4 and G6

yields S[lm(n-1)]=0 and S[lm(k-1)]=0. C5a now implies that lm(n)=1 and lm(k)=1. But the system is then incompatible with C5a. Thus, the system has no feasible solution, and the protocol cannot reach a state in which it will halt. If we eliminate the tokens involved with error recovery by setting the following constraints: S[r(n)]=1, S[r(n+1)]=1, S[lm(n-1)]=0, S[lm(n)]=0, S[rr(n-2)]=0, and S[rr(n-1)]=0, then equation C5a becomes

$$S[wm(n+1)] = S[msg(n)] + S[msg(n+1)] = 1.$$

Condition H3 shows immediately that the system is once again infeasible. Thus, when the error recovery messages are removed and no messages are lost (S[lm(n)]=0) the protocol cannot reach a state in which it will halt. This demonstrates that the protocol behaves as expected. The reduction to an infeasible form can be handled in an automatic way, if desired, by applying techniques from the SIMPLEX algorithm of linear programming.

SUMMARY

This analysis demonstrates that the protocol behaves as expected. The relationship between states and messages is preserved regardless of the order in which events occur. The process state always reflects the transmitted messages for which responses have not been received. Since the prototype implementation of the protocol is derived directly from the model and functions as a state machine, we expect that it will exhibit the same properties as the protocol model. Testing of the prototype validates the predictions of the model analysis.

REFERENCES

1. Farber, D.J. and K.C. Larson, "Network Security via Dynamic Process Renaming," Proceedings of the Fourth Data Communications Symposium, (October 1975).
2. Larson, K.C., "A Prototype Implementation of a Protocol for "etwork Security," Technical Report No. 84, Department of Information and Computer Science, University of California, Irvine, (May 1, 1976).